Task Computing
- the Semantic Web meets Pervasive Computing -

Ryusuke Masuoka[1], Bijan Parsia[2] and Yannis Labrou[1]

[1] Fujitsu Laboratories of America, Inc.,
8400 Baltimore Avenue, Suite 302, College Park, MD 20740, USA
{rmasuoka,yannis}@fla.fujitsu.com
http://www.flacp.fujitsulabs.com/
[2] MIND Lab, University of Maryland,
8400 Baltimore Avenue, Suite 200, College Park, MD 20740, USA
bparsia@isr.umd.edu
http://www.mindswap.org/

Abstract. Non-expert users have to accomplish non-trivial tasks in application and device-rich computing environments. The increasing complexity of such environments is detrimental to user productivity (and occasionally, sanity). We propose to reduce these difficulties by shifting focus to what users want to do (i.e., on the tasks at hand) rather than on the specific means for doing those tasks. We call this shift in focus "task computing"; we argue that "task computing" offers an incentive to device manufacturers to incorporate semantic web technologies into their devices in order to get the benefits of easier and more flexible use of their devices' features by end-users.
To support task computing, we developed an environment called a "Task Computing Environment" (TCE), which we have implemented using standard Semantic Web (RDF, OWL, DAML-S), Web Services (SOAP, WSDL) and pervasive computing (UPnP) technologies, we describe and evaluate our TCE implementation, and we discuss how it has been used to realize various device-usage scenarios.

1 Introduction

Fujitsu is an Information Technology company with a business that spans the entire range from chip design and manufacturing to integrated IT solutions. As a device manufacturer, Fujitsu is interested not only in designing and marketing devices (of various kinds) that deliver more functionality in each new product cycle but also in making this functionality user-accessible and incorporating it in higher value, integrated IT solutions. We discuss an application of Semantic Web and Web Services' technologies that supports these goals shared by many device manufacturers and IT solution providers.

Being knowledge workers ourselves we face the frustrations of trying to do simple things in the device and application-rich environments where we work. Let us consider the rituals of visiting a lab or a business unit in order to make a presentation and discuss collaboration with colleagues. We exchange printed business cards that end up in a desk drawer because we never have the time to enter them into our PIM, we have to deal with capricious projectors that might or might not work as intended while we try

to make a presentation, we promise to e-mail a soft copy of our presentation when we get back to our office or to follow-up with a new scheduled meeting, we search for addresses and directions to the restaurant for the business dinner or to the airport and print them on paper to take with us. Performing each of these simple tasks entails a complex choreography of interaction with a variety of devices and applications, cutting and pasting and a lot of time and frustration. The goal of the discussed work is to make it easy for the average user (who typically is neither the authors nor the readers of this paper) to perform each of these tasks with a few clicks; by focusing on the tasks that the user wants to accomplish, rather than the process of accomplishing them; we refer to our approach as Task Computing.

Several features of Window XP exhibit a very limited form of task orientation. For example, when the user inserts a music CD to the CD tray, a window pops up suggesting to the user tasks she can perform from that point on, such as "Play CD", or "Copy Music". In this case, the operating system uses a pre-specified list of actions, or tasks, that is associated with the occurrence of a specific event (i.e., inserting a music CD), so that when the event occurs, the relevant listing of actions is presented to the user to act upon. In Microsoft Office XP, the smart tags feature works similarly; it highlights text in the current document (while using Microsoft Word) and offers the user a drop down menu of actions that can be performed with the object that that text denotes. For example, if the text represents a name, then this feature may identify the object associated with that name to be a person, and may offer to send mail (to that person) or schedule a meeting (with that person). Microsoft XP and Microsoft Office are not the only examples of attempts to present to the user a more task-oriented view of the computing environment. When a user types an address in the search box of Google, the service will return (above the usual search results) a link to Yahoo maps or MapQuest that, if followed, will provide a map of the address.

These task-oriented systems exhibit three fundamental limitations:

- The functionality has been designed into the application; the application's programmers have programmed (hard-wired) the system's response.
- The system fragilely relies on correctly guessing the meaning of the input by its syntactic features.
- The system employs a cause-effect (or trigger-response) mechanism, in the sense that a certain type of input results to a single action (application invocation). There is no support for more complex user-composed and initiated workflows.

Task computing, as defined and elaborated in this paper, is an attempt to address these limitations. But we also believe that Task computing is an important step towards the realization of the pervasive computing vision [1] by enabling non-technical users to accomplish complex tasks in device-rich and service-rich pervasive computing environments. In the remainder of this paper we first elaborate on the notion of Task Computing and the set of technologies we have used in order to enable it. We discuss applications powered by Task Computing (specifically, the scenarios mentioned earlier in this section) and we present in detail our implementation of a Task Computing environment. We proceed to evaluate our accomplishment thus far and to discuss them in light of related work and we conclude with some of the future directions for this work.

2 Enabling Task Computing for Pervasive Environments using Semantic Web technologies

We define *Task Computing* as computation to fill the gap between tasks (what user wants to be done), and services (functionalities that are available to the user).

Task computing seeks to redefine how users interact with and use computing environments. We are particularly interested in pervasive computing environments, which typically involve multiple devices and services (functionalities) and non-technical users, with no a priori knowledge of the environment, operating mobile devices with limited interfaces. fundamental premise of Task Computing is to present to the user the tasks that are possible in the user's current context, to assist the user in creating more complex tasks by using simpler tasks as building blocks and to guide the user through the execution of the complex tasks. Once complex tasks are defined by the user (or perhaps by other users) such tasks can be re-used, very much like macros in popular applications today. Unlike macros (at least the ones that Microsoft Office users are familiar with), the building block tasks and the resulting complex tasks may span multiple applications and multiple computing platforms (not a single device). In fact, one of the goals of Task Computing is that the user does not need to know or care about how the actual constituent tasks are executed or where (on which machine) they might be executed on. The ultimate objective of Task Computing is to present to the user an abstract view of resources (devices and services) that can be used in an ad hoc manner by the user, in order to execute tasks of arbitrary complexity

2.1 Task Computing Framework (TCF) and Task Computing Environment (TCE) definitions

A Task Computing Framework (TCF) is a framework that supports Task Computing, by providing support for:

- The workflows of Task Computing, i.e., at a minimum, Discovery, followed by Composition and Execution
- Semantic description of tasks and services
- Specification, Execution and Re-Usability of tasks by end-users
- Manipulation including creation and disposal of services by the end-users

Our definition of a Task Computing Framework does not make reference to computational components. We will refer to an embodiment of a Task Computing Framework as a Task Computing Environment (TCE). A TCE is a computational system that includes, at a minimum, the following components.

- One or more Task Computing Clients (TCCs),
- One or more Semantically Described Services (SDSs),
- One or more Semantic Service Discovery Mechanisms (SSDMs), and
- Optionally one or more Service Controls (SCs)

Later, we present in detail a Task Computing Client embodiment, called Semantic Task Execution EditoR (STEER), embodiments of multiple Semantically Described Services and Service Controls.

2.2 Technologies and challenges of designing a TCE

At the technical level, Task Computing relies on the on-the-fly discovery, selection, real-time (as opposed to design time) composition of services and execution of such compositions; the compositions represent end-users tasks. While any TCE ought to support these capabilities, and there are many ways to do so, in our TCE we chose to rely on Semantic Web Services technologies, in particular, DAML-S, OWL and Web Services technologies, specifically WSDL. While these choices addressed various issues facing our implementation (as described in Section 4), they introduced their own difficulties.

In assembling a TCE we faced multiple challenges:

- We had to assemble service discovery, composition and execution in a comprehensive framework where the selected technical choices needed to be mutually compatible. While the various Semantic Web/Web services standards went a long way toward interoperability, the standards and standard implementations thereof were in varying states of completeness. Over the course of developing our TCE, these standards changed and their implementations became more mature.
- We confronted the gap between the semantic layer of services and that of their execution.
- We had to design a user interface targeted to relatively unsophisticated end-users which would also work well and consistently across various devices and operating systems.
- We needed a plausible ontology translation mechanism to manage both inter-device-ontology mapping and the proliferation of services and tasks which use them. We had to develop techniques for managing highly dynamic sets of services, where many of the available services were created, advertised, hidden, or published by end users on an *ad hoc* basis.

Many difficulties were simply eliminated by our choice of Semantic Web and Web services technologies. For example, interlanguage function calls, even across the Internet, is trivial with WSDL described, SOAP based APIs. Web services provide a uniform interface for invoking program functionality across implementation and execution systems, whether these be language runtimes or actual computers. We currently use SOAP (in particular, the HTTP/RPC binding) as our wire protocol and direct invocation mechanism for all services. This approach provides language, operating system, and machine independence. We also use the Web Services Description Language (WSDL) for low level details of the service API. Both of these technologies provide excellent support for programmers and system integrators, but, by themselves, are not particularly suited for end users. Task computing, as we have presented it, relies extensively on service discovery and composition. The sorts of solutions for these needs offered by the Web services community (e.g., UDDI for discovery and BEPL4WS for composition) are similarly programmer oriented. These latter technologies are also less widely supported and are the focus of much less industry consensus.

3 Applications

Although we envision that Task Computing will gradually replace current personal computing experiences (those on desktop and laptop computers) as the main way to interact with computing devices, we currently focus on application environments where mobile users need to interact with transient devices and services. Examples of such environments include office spaces, home entertainment and healthcare.

Here we describe task computing application scenarios in office environment in favor of perspicuity and familiarity of the readers. The user is a salesperson (let us call her Alice) that meets Bob and his group for the first time in their conference room. Alice is allowed to wirelessly connect to the network provided for visitors. Alice's STEER (Semantic Task Execution EditoR, an implementation of TCC. See Section 4.1) on her PDA discovers many services in the room which can be used for interesting add-hoc compositions that can be executed in real time, via the Compose page of STEER.

Exchanging Business Cards

STEER infers a permissible composition of "Contact Provider" service for Bob running on his PDA and "Add into Contact List" on Alice's PDA. Alice hits the "Execute" button to execute [Contact Provider (on Bob's PDA) + Add into Contact List] to add Bob's contact into her PIM. Specifically, "Contact Provider" service for Bob is first executed and STEER acquires Bob's contact. Then STEER executes "Add into Contact List" running locally on Alice's PDA.

Showing and Sharing the Presentation

Alice is about to start the presentation of her product; the presentation file is stored on her PDA. Alice uses the composition of "Local File" service on her PDA and "View on Projector" service in the conference room [Local File + View on Projector]. As a result, Alice's presentation is projected on the screen and the control page shows up on her PDA. Using the control page, she can control her presentation wirelessly. Bob asks Alice for a copy of her presentation. Alice executes [Local File + Bank (File)] thus making the presentation file available through the "Bank (File)" service in the conference room. When someone later comes in the room, she can find the file to project it to the projector or copy it into her computing device even after Alice leaves the room.

Scheduling a Future Presentation

Bob and Alice schedule a future meeting in the same room. Bob creates the schedule item in his PIM on his PDA. He then composes and executes [Schedule from Scheduler + Bank (Schedule)] thus making the schedule item available as a service in the conference room environment. Alice can then execute [Schedule Provider (created by Bank (Schedule)) + Add into Schedule List] in order to add the newly scheduled future presentation into her PIM; other attendees can do the same, either now or later since the service will remain available in the room even after Bob leaves the room.

Checking and Printing Directions to the Airport

Alice asks Bob about how to get to the airport and Bob suggests that she use her STEER to see and print the directions. A map route service that displays on a public

display the route between the office and other destinations is available in the conference room, along with services that provide the addresses of the nearby airports. Alice proceeds to perform a composition the [Map Router + Address of BWI airport], which when executed results to the route from the office to the BWI airport being displayed on the "Map Router" service's display. Alice can manipulate the display (zoom-in, zoom-out, pan, etc.) from her own PDA and she can even print the map route and directions.

It is important to note that each of these tasks can be performed in more than one ways. The services we have built can be deployed in any other environment and used by a user's TCE in that environment. Needless to mention that there are ways to achieve these tasks with today's technologies. What is unique though, is that none of the individual services was designed with these tasks in mind, that the existing services (40-50 in our demonstration environment) can be combined in a multitude of ways to perform interesting tasks (of which we only mention a handful) and finally, that the user need not discover, or familiarize themselves with the available services, in advance.

4 A TCE Embodiment

As mentioned in Section 2.1, a TCE consists of four components. In implementing each component, and assembling the components into a coherent whole, we have tried, as much as possible, to use Web standards with commodity like available implementations. In this, we were largely successful, modulo the relative immaturity of some of the standards we selected. This suggests that a reasonable TCE does not require special infrastructure support, but can be implemented almost entirely at the application level. This allowed us to make the various components self-contained, loosely coupled, and flexibly configurable and our TCE to be rapidly deployed in ad-hoc network environments. So, for example, our TCE still offers added value when dealing with plain UPnP services lacking rich semantic descriptions. Figure 1 shows the global architecture of our implemented TCE. Of course, this is just one of many possible TCEs.

4.1 STEER – Task Computing Client (TCC)

In this section, we describe how Semantic Task Execution EditoR (STEER), a Task Computing Client (TCC) works based on "Showing the Presentation" scenario described in Section 3. STEER offers a user interface for presenting to the user what she can do in the current context, and lets her define and execute the tasks.
We will attempt to describe the user experience of interacting with a TCE and simultaneously describe how the system accomplishes the user experience. Consider yourself to be a first time visitor to a company office. You are taken to the conference room and you get on the network provided for the visitors in the room. Then you want to give a presentation using the file on your computer. After you get the access to the network, you see a page in your Web browser on your computer like the one in the left

side of Figure 2. This is the STEER Compose page, which shows the two-step service compositions as tasks you can do in this environment. This happens as follows.

First there is a discovery phase. STEER searches for the local and pervasive services[1] available and keep checking newly added or removed services.
STEER consults the local service manager (see Figure 1) for the available local semantic services such as "Local File" service which lets you choose a local file and expose it as a Web page. For each of those local services, STEER retrieves its Semantic Service Description (SSD) in DAML-S [3] from the local file system and feeds it into its inference engine (IE).

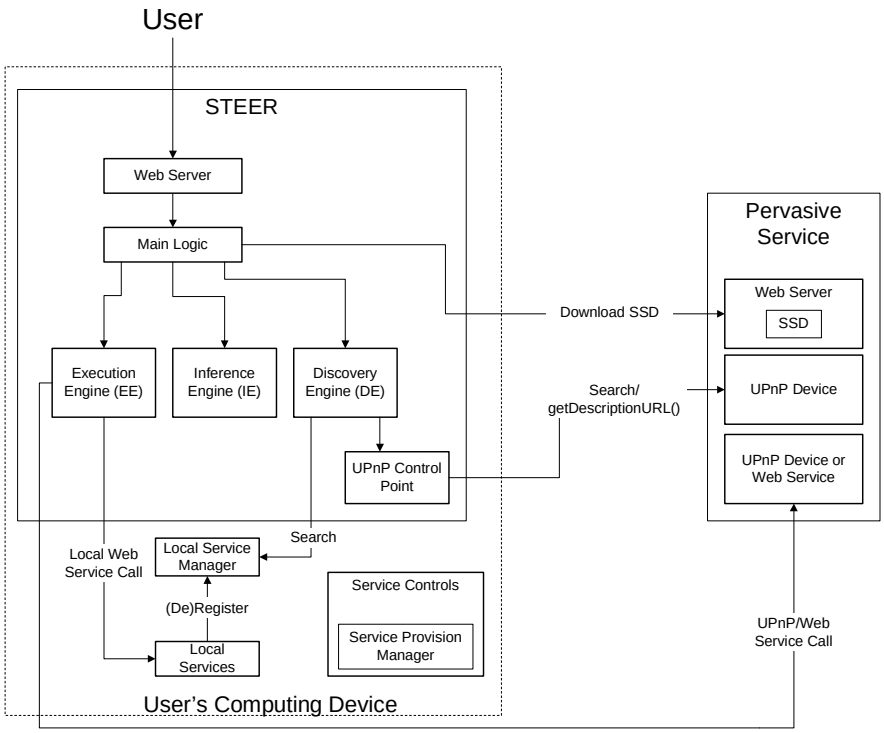

Fig. 1. Architecture of Task Computing Environment: User's computing device such as laptop and PDA runs STEER, a TCC, local service manager and optionally local services and service controls. Service controls may be running on other computers. A pervasive service consists of Web Server, UPnP device, and optionally Web Service. More details are given in the text.

[1] We define the local services as the services running on the user's computing device and private to the device. Pervasive services are defined as the services on the immediate network found through network service discovery mechanisms such as UPnP.

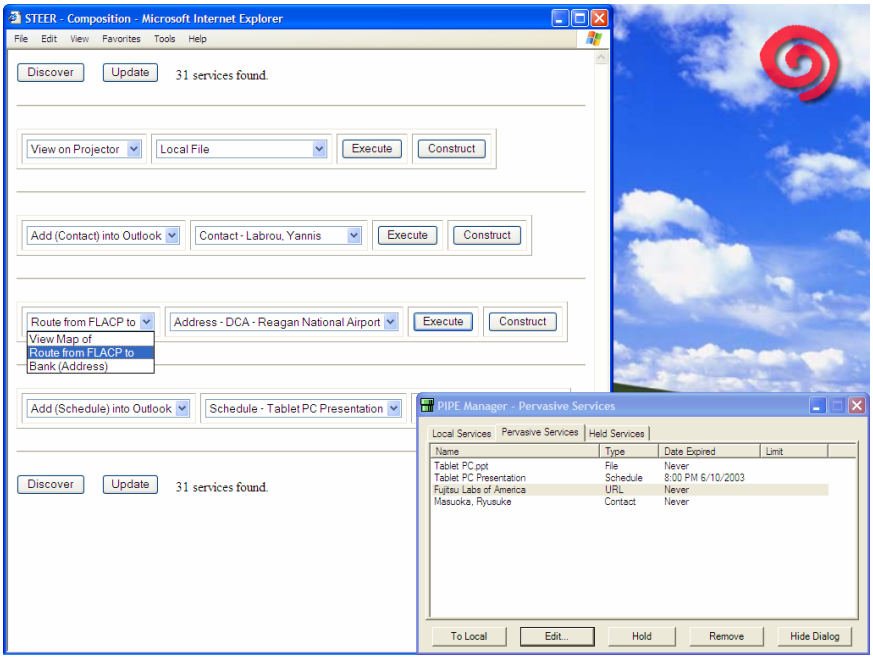

Fig. 2. Screenshot of Task Computing Environment (TCE) Client Desktop: STEER is a Web-server based TCC. STEER's Compose page is shown in the Web browser on the left. The compositions are categorized based on their semantic inputs and outputs. In each pair, any service on the right side drop-down menu can be combined with any service in the left side drop-down menu. The user uses the drop-down menu to choose the service and "Execute" button to execute the composition. The user can click "Construct" button to create a more complex composition starting from the two-service composition in this Compose page. The swirl on the upper right corner is the "White Hole." The user can drag and drop objects such as files, URLs, contacts and schedules from PIM to create services that provide semantic versions of the objects dropped into it. After the drag and drop, it swirls for a while to let the user know it is successfully dropped. Finally on the lower right corner a management GUI is shown; the user can use it to manage the services. Through this GUI, the user can make the services local or pervasive or hold temporarily those services for possible future use. The user can also control how those services are provided such as expiration time, etc.

STEER searches the services on the network using UPnP [2] device discovery protocol.[2] The UPnP architecture offers pervasive peer-to-peer network connectivity of devices and PCs. Pervasive services such as "View on Projector" respond to the search message as UPnP devices and are found by STEER. For each of those found UPnP devices, STEER makes a specific UPnP call (getDescriptionURL()) into it to get the URL of its SSD (expressed in DAML-S). STEER determines the service as a

[2] STEER acts as a control point and while pervasive services act as UPnP devices in UPnP terminology.

semantic service if it gets a result for this UPnP call. Then, STEER downloads its SSD in DAML-S from the URL returned. STEER feeds the SSD into its IE.

Then there is a composition phase. When STEER is asked for the Compose page, STEER asks its IE to compute prioritized two-step compositions starting with no-input services based on input and output matching. The order of compositions is prioritized based on various metrics including recentness of the service discovery and the exactness of matches. The IE returns possible compositions that are used by STEER to create the Web page which appears on the left in the Web browser in Figure 2. The Web page is dynamically created entirely from SSDs of found services. The menus are categorized based on the semantic objects handed between the services, such as "Web Page," contact, schedule, and address. The combination of "View on Projector" and "Local File" in the line for "Web Page" category is the composition you want to execute.[3]

Two-step composition starting with a no-input service has a couple of merits including: (1) The composition is always immediately executable with no input necessary to start, (2) Compositions allows interpretation as English sentences such as "View on Projector, Local File," and (3) Two-step composition, assuming that the constituent services are of the right granularity, are easy for the user to grasp.

Now you can immediately execute the composition by clicking the "Execute" button next to the particular composition. Or you can change the service to be executed, with any other compatible service by using drop-down menus. You can even construct longer compositions starting from any given two-step composition by clicking the "Construct" button in the same line as the composition. That leads you to the Construct page (figure not given in this paper) where you can modify the composition or add more services to the composition.

Let us assume that you decide to execute the service composition of "View on Projector" and "Local File" and click the "Execute" button. Figure 3 explains what happens when you click the "Execute" button. The Execution Engine of STEER (see Figure 1) handles the execution of the composition and makes the UPnP and Web Service call. Since the composition happens in the semantic layer, STEER needs to bridge the gap between the semantic objects in the semantic layer and parameters of and results from UPnP and Web Services. Using the grounding information in the SSDs of the services, STEER unmarshals the return values from UPnP and Web Service calls to the semantic object and marshals the semantic object to the parameters for those calls. This grounding mechanism, which happens in the Execution Engine, allows STEER to operate in the semantic layer and still to be able to execute all UPnP and Web services.

When the first service, "Local File" is executed, the service pops up a dialog box to let you to choose a file from your local file system. When you finish with the selection of the file, the service copies the file into a Web folder and returns the URL to the copied file as a string return value. The string is then unmarshaled into a "Web Page" semantic object by STEER internally.

---

[3] Note that "Local File" service, which has no-input, is the first service to be executed and the "View on Projector" is the second one to be executed.

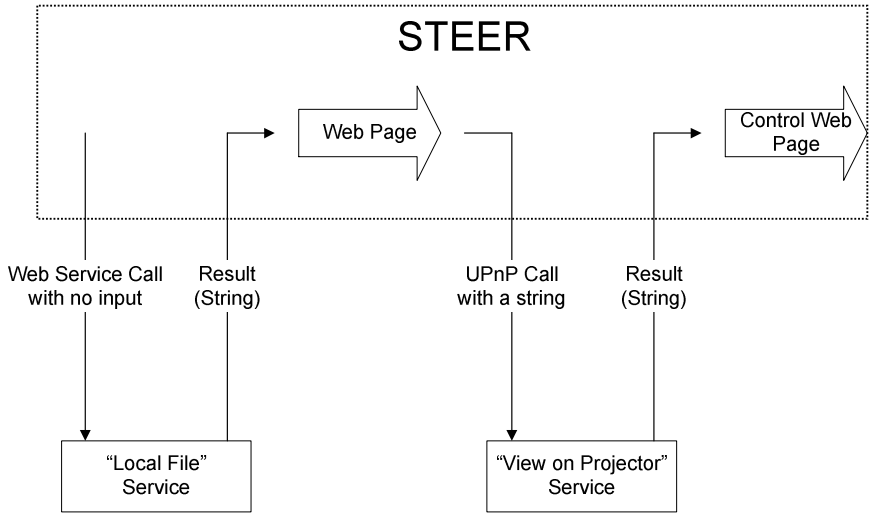

Fig. 3. Execution of the services by STEER: STEER uses the grounding information in their Semantic Service Descriptions (SSDs) to execute the services. In this case, the "Local File" service (which is a local service) is invoked through (local) Web Service call with no input based on WSDL/SOAP. The return value is just a string of URL, then STEER unmarshals it into a semantic object (which is "Web Page") using the grounding information in the SSD of "Local File" service. Then the "Web Page" semantic object is passed to the next service ("View on Projector") and marshaled into the string parameter for UPnP call. The same procedures apply if there are more services to follow in the composition to be executed.

The second service, "View on Projector" is executed with a string which is marshaled from the "Web Page" semantic object from the last service. "View on Projector" downloads the page from the given URL and displays it on the projector. Thus the execution of the composition enables you to show the file originally on your local disk on the projector in the room.[4]

An important point to note is that there has been no prior arrangement of those services and no knowledge of each other in advance. Both services happened to be found by STEER in a particular context, dynamically composed, and executed in a composition to accomplish a task.

Now let us turn to the issues of who and how to create those SSDs that makes it possible. The semantic object, "Web Page" in Figure 3 happens to be relatively simple with just one property of URL string. But STEER also needs to deal with other more complex semantic objects like address, schedule, and contact. While the grounding

[4] You can actually control the "View on Projector" service so that you can give a decent presentation. This is possible since STEER automatically appends and executes "View Locally" service at the end of composition if the composition ends with a "Web Page" semantic object. The composition in this case ends with a "Control Web Page" to control "View on Projector" service. "View Locally" displays the "Control Web Page" (a subclass of "Web Page") in the Web browser locally on your computer.

mechanism of marshalling and unmarshalling between semantic objects and parameters for the service invocations stays the same whether they are complex or simple, creating the grounding information for the complex semantic objects is substantially more difficult. Though we do not expect end-users of Task Computing create the grounding information, we foresee IT professionals at the device manufacturers and IT departments of large organizations will start to create the semantic service descriptions (SSDs) with grounding information for their devices and Web services.

We have created a tool called "OntoLink" to help such IT professionals to generate SSDs. "OntoLink" provides its user a GUI, which lets him visually create the SSD complete with the grounding of the semantic object in an ontology to the WSDL of the Web Service. Currently OntoLink supports grounding by XSLT scripts. The Execution Engine uses those XSLT scripts in the execution of services.

OntoLink also supports mapping (a kind of ontology translation) creation between the semantic objects in different ontologies. Grounding allows STEER to operate in the semantic layer, but it does not solve all the interoperability problems. Even though we expect that ontologies used in pervasive environment will be controlled mainly by device and software manufactures or consortia of such players and subsequent oligopoly of such ontologies, we will still need to have mapping between the semantic objects to freely compose the services which happened to be in different ontologies and many remote Web services. We see the efforts of creating the mappings fall on IT professionals as well. "OntoLink" lets its user open in its GUI two ontologies, which include the objects to be mapped. Through visual manipulation, the user produces the SSD of the mapping service.[5] Currently OntoLink supports semantic object mapping by XSLT scripts.

4.2 Services and Service Controls

In building up our TCE, we needed a reasonable set of "right-sized" services that would mesh appropriately with the sorts of objectives the particular set of end-users we had in mind (office workers) would have. Just as device manufacturers have to find the right grouping of functionalities to present to their users, so we had to provide services that were neither too packaged, nor too fine grained to fit in with our user's expectations and understanding. We had a further constraint of needing many services that made sense in no-input, two-step compositions.

In designing our services, we took inspiration from menu items commonly supplied across many sorts of office applications. Short term, adoption of TCEs will be by users already experienced with current applications, so it makes sense to provide them with familiar functionalities. Long term, it is clear that such ubiquitous presentation of functionality represents some degree of consensus understanding of the task in computer oriented offices.

We have implemented more than twenty kinds of services. They are categorized as follows:

---

[5] Though we do not go into the details in this paper, we treat semantic object mappings also as services and this decision made the whole architecture very consistent.

- Instance Selector: Lets the user select an item from a local system (file system, PIM, etc.) and outputs it as a semantic instance for other service to consume. This category includes such services as “Local File” and “Schedule from Scheduler.”
- Instance Copier: Lets the user copy the semantic instance into a local system. This category includes such services as “Add into Contact List” and “Add into Scheduler.”
- Instance Provider: Provides a semantic instance for other service to consume. This category includes such services as “Contact Provider” and “Schedule Provider.”
- Instance Handler: Handles the semantic instance. This category includes “View on Projector” service as “Web Page” handler. “Map Router” service as “Address” handler, etc.

As in the right side of Figure 1, pervasive services have architecture with an UPnP device, a Web server, and optionally a Web service. UPnP device supports its discovery and returns the URL of its Semantic Service Description (SSD). The service provides the SSD by itself from its own Web server. The service which the SSD exposes is provided from its UPnP device or Web Service. One difficulty we needed to overcome is that its SSD may need to be created dynamically to accommodate the changes of environment the device happens to be in.

Other denizens of TCE are service controls. Service controls let you manage the services. You can create, hold temporarily, remove, or change how the services are provided.

In order to illustrate how the service controls are used, we describe how a business card exchange scenario works in TCE.[6] When you meet someone and want to give your business card to her, “White Hole” service control enables you to do so (see Figure 2). You first drag and drop your contact item from your PIM into the White Hole. Then the White Hole parses the contact item and creates a “Contact Provider” service for your contact. She finds this “Contact Provider” service and executes the composition of “Add into Contact List” and “Contact Provider” to insert your contact into his PIM. In the same way, you can share with her any other contact information as long as you have the contact in your PIM. By default, “White Hole” is configured to remove the service it created after a certain period of time, but you can remove it immediately or you can keep it forever by using the management GUI (see the lower right corner of Figure 2) if necessary.

With service controls, you can change the environment (which is defined here as the set of services available in your proximity) proactively for others as well as yourself. These service controls allows you to use the environment as communication medium. “Semantic Instance Bank” service controls let you to leave instances in the environment.[7] As in the “Scheduling a Future Presentation” scenario in Section 3, you can leave a schedule instance for the meeting in the conference room where the meeting will be held. The person who comes in the room later can find the schedule and insert it into her PIM if necessary.

---

[6] This is only one way to execute the “Exchanging Business Cards” scenario in Section 3 .

[7] “Instance Providing” services to be exact.

The current list of implemented service controls includes:

- White Hole: Lets the user drag and drop an item (file, contact, schedule, etc.) from local systems (file system, PIM, etc.) to create (and manage) a corresponding instance providing service
- Semantic Instance Bank: Semantic service for each type of semantic instances to create an instance providing service
- Image Service Control: Lets the user interact with image devices (scanner, DV/digital camera) and create an instance providing service for the image file captured.

We need to clearly state that this is all built already and that it works. We have installers for the complete TCE, so that a user can install the client environment on her machine in a couple of minutes and use it immediately once in a TCE environment.

5 Evaluation

In building this TCE, we wanted to build an end-to-end system, from discovery through composition to execution, all actually using the relevant technologies. This was often difficult as the maturity of the libraries (e.g., WSDL) or even the standards left a lot to be desired. We often traded "vertical" for "horizontal" completeness. For example, STEER only supports compositions of services with at most one input and/or output into sequences aligning those inputs and outputs. Thus, our DAML-S Process Model interpreter doesn't execute the more complex DAML-S control constructs, such as if-then-else, iteration, or split. However, given our target scenario – office workers without a lot of computer expertise – this limitation seemed not only acceptable, but desirable. It's unclear how to make more complex composing behavior feasible for our target audience.

Similarly, our reasoner isn't complete with regard to DAML+OIL semantics (much less supporting various useful, non-standard inferences), but given the relative simplicity of our matchmaking and selection needs, being lightweight and flexible trumped other features. Indeed, the success of such ad hoc reasoning services validates our framework. Often with component and assisted programming environments, there is difficulty in scaling down the implementation (and user) requirements. Just as regular programmers can build useful and interesting content management systems using Web standards and common components such as Apache, MySQL, and the like, we expect that regular programmers and system administrators will be able to build custom TCEs tuned to their particular situation and driven by their interests and needs.

Clearly, other scenarios would drive rather different design tradeoffs and thus rather different TCEs. However, it's pretty clear how to build those alternative TCEs out of ours. The easiest thing to change is the set of available services. As long as they can be cast into a single input/output model, our TCE will adapt automatically. But even if, say, in a Grid computing scenario, or a complex publishing one, the services supplied and the workflows desired were more elaborate, TCEs designed to handle such elaborations could easily reuse our discovery mechanisms, executor, inference

engine (perhaps slightly extended), and even many of the services and meta-services (e.g., Service controls such as “White Hole” and “Semantic Instance Banks”).

In general, TCEs are driven by the ordinary behavior of regular users focused on the job at hand. In this regard, one serious limitation in our current implementation is the lack, at least at the STEER UI level, of the ability to save and share compositions and generalized composition templates. We do have TCE clients that can do this, but we have not integrated that functionality into the overall system. The accretion of such embodied expertise in the system is critical to making the system truly dynamic. Merely having flexible, parameterized services is not enough, as they just generalize the predetermined set of actions somewhat. For a TCE to be user driven, it must allow for, support, indeed encourage the seamless expansion of the set of available, salient, task “conscious” services.

6 Related Work

Our work builds heavily on the convergence of the Semantic Web and Web services, particularly as supported by the DAML-S framework [3]. We use the DAML-S Profile upper ontology to represent and categorize our services for matchmaking and in-composition selection of services. All compositions are represented using the DAML-S process model, and executed directly by a process model interpreter. We extended the grounding ontology to directly handle UPnP, as well as providing support for marshalling and unmarshalling between XML schema and DAML+OIL. The latter extensions have been including in the lastest version of DAML-S.

Bussler, Maedche, and Fensel [4] suggest that, unlike the Semantic Web enabled Web Services (SWWS) conceptual architecture they propose, DAML-S doesn’t present a “a mediated P2P environment” which requires every “trading partner to be able to mediate differences in document types, semantics or even business process behavior.” To address this requirement, their architecture contains a “semantic transformation engine” as a critical component providing “ontology based mediation.” Our TCE incorporates exactly such a mechanism without having to particularly extend the DAML-S framework. Our translators are just more services, albeit services of a specific type that a translation aware matching engine can make special use of. But this is no different than other classifications of services (in house vs. subcontracted, local vs. remote, UI vs. processing, information providing vs. world altering, etc.) that we might need to adopt for a variety of purposes. An advantage of our DAML-S based approached is that the system degrades smoothly when using more generic tools. Since ontology translators and mediators are just more services, a perfectly generic composer can use them.

Unlike much of the current work on “semantic matchmaking” (particularly in a DAML-S context), our approaches did not rely on either a broker or other repository mechanisms [7] or a sophisticated P2P scheme [4]. Given the limits imposed by the pervasive and local contexts, and the fact that we had only one fundamental scenario implemented, having every service announce itself to the client’s knowledge base was quite feasible. The key discovery activities became selection of services in a particular

composition context [5]. As we develop TCEs for more heterogeneous environments (e.g., where the user is moving from boardroom to coffee shop), we envision using more elaborate pervasive brokering schemes, such as described in [8]. Similarly, as remote services start to proliferate, we will need to find some way of finding and filtering them. We expect to have to support a number of mechanisms from enhanced UDDI repositories [7], to user bookmarks, to RSS feeds, to a service aware Google, etc.

7 Conclusions and Future Work

We believe that Task Computing is a promising application of Semantic Web and Web Services technologies, especially in pervasive computing environment. What is unique about our Task Computing implementation is that we have implemented end-to-end system from discovery, to composition, and eventually execution of services, all of this driven by end-users, rather than by developers, via a user interface. In order to realize such a system and let the user define tasks at the semantic layer, we had to bridge the raw (in the sense of semantics) APIs that drive the execution layer and the semantic layer (the user-facing side) at all layers of the system (discovery, composition and eventually execution of composed services). But when completed we were able to provide the users with a task-oriented view of their environment that in turn minimizes the required knowledge and skills for accomplishing interesting tasks in that environment.

We see Task Computing as a business opportunity for both device manufacturers and IT solution providers. For device manufacturers, devices can be treated as Semantically Described Services (SSDs); someone will have to provide this semantic layer wrapper of the devices' functionality. In addition, the devices need to implement a service discovery mechanism (e.g., UPnP or Bluetooth) and make available a remote control API for accessing their functionality (e.g., UPnP). TCE clients such as STEER can be pre-configured with SSDs for well-known devices; or some web server in the network can provide SSDs for well-known devices; in other words, the provider of the semantic layer need to be the device manufacturer itself. Device manufactures will benefit from the newly found uses of their devices in flexible and ad-hoc Task Computing environment.

IT solution providers can combine the benefit of Task Computing-enabled devices with additional services, to provide interesting application is various verticals in an easier and faster manner than today. In a way, since the end-user becomes the application builder, the solution provider need only focus on the set of services to make available in the vertical environment and the user interface for that environment. The first task for the solution provider would be to select the web services, some of them, third party services that they want to make available to end users and create the semantic descriptions and grounding descriptions for them. Tools like OntoLink will help them with that task. The tangible advantage is not only the faster development cycle but the ease of maintenance and of adding functionality into the system, since they only need

they only need to add new services or to change the grounding of an existing service, in order to “upgrade” a service implementation.

Our system is the first version of a vision; as a result there are a lot of issues that we have barely considered thus far. Trust and security mechanisms, monitored execution of asynchronous, constantly-changing, or subscription services, deeper integration with location determination and other modalities for user interface such as voice, are just a few of the directions to further explore. Nevertheless, we believe that Task Computing has the potential to change how the end-users interact with their computing environments.

Acknowledgments

We want to thank Jim Hendler for his guidance and constructive criticism throughout this project and the writing of this paper. We also want to thank Mike Grove, Zhexuan Song, Wei-Lun Chen, Evren Sirin, and Aditya Kalyanpur for their hard work to make the Task Computing vision a reality.

References

1. Weiser, M.: The Computer of the 21st Century. Scientific American, vol. 265, no. 3, Sept. 1991, pp. 66-75
2. UPnP Forum: http://www.upnp.org/
3. The DAML Services Coalition (alphabetically Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara), "DAML-S: Web Service Description for the Semantic Web", The First International Semantic Web Conference (ISWC) , Sardinia (Italy), June, 2002.
4. Christoph Bussler, Alexander Maedche, Dieter Fensel, "A Conceptual Architecture for Semantic Web Enabled Web Services", ACM Special Interest Group on Management of Data: Volume 31, Number 4, Dec 2002
5. Evren Sirin, James Hendler, Bijan Parsia, "Semi-automatic Composition of Web Serv-ices using Semantic Descriptions", "Web Services: Modeling, Architecture and Infrastructure" workshop in conjunction with ICEIS2003, 2002
6. A. Flett. A comparison of daml-s and wsmf. In Internal Report, Vrije Universiteit Amsterdam, 2002.
7. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara, "Semantic Matching of Web Services Capabilities", The First International Semantic Web Conference (ISWC), Sardinia (Italy), June, 2002.
8. Harry Chen, Tim Finin, and Anupam Joshi, Using OWL in a Pervasive Computing Broker, Workshop on Ontologies in Open Agent Systems, AAMAS 2003.